Stat 238, Fall 2025
Project Minis

Alexander Strang
Due: by **5:00 pm** Friday, March 21st, 2025

Mini Project 3: Bayes Optimal Coding, Search, and Experimental Design

Policies

- You may work with up to **1 partner**. To find partners, talk to your friends in class, or post on the Ed thread associated to this project mini.
- All submissions must be properly type-set and sourced.
- Project minis should be uploaded to Gradescope under the appropriate mini-assignment. If you work in a group (eligible projects are marked), you should submit one assignment, and link your partners. Please do not submit duplicated assignments.
- Please submit a single pdf. Properly export any notebook or code files to pdf so that their contents fit within the assigned pages.

Background

In class, we've seen a few examples of Bayesian decisions problems where we aim to select an optimal measurement scheme. These problems arise frequently in optimal experimental design (what experiment should we run?), search (what sequence of questions should we ask?), and coding (how can we optimally compress the codes used to represent a set of possible objects?). These problems are all related by a standard framing.

Imagine that you are asked to resolve an unknown Θ drawn from a prior distribution p_Θ. You have access to a set of possible measurements, $\mathcal{M}$, that you could make. Let $m \in \mathcal{M}$ denote a possible measurement. For experimental design, we might use $\mathcal{E}$ for a set of possible experiments we could run, and for search we might use $\mathcal{Q}$ for a set of possible questions we could ask. Since this assignment will mostly work with categorical unknowns drawn from a finite set, we will adopt $\mathcal{Q}$ for the set of possible questions we could ask, and $q \in \mathcal{Q}$ for a particular question.

If you select a particular question to ask, q, then you recieve back an answer $A \sim p_{A|\theta,q}$ where θ is the true value of the unknown, q is the question asked, and $p_{A|\theta,q}$ is the sampling distribution for the answer given the question and the true unknown. The sampling distribution fixes our likelihood function. Each distinct question we could ask produces a distinct likelihood function.

When you receive an answer, $A = a$, you condition on it to update your beliefs regarding the unknown. By combining the information you observed, a, and your prior over the unknown, you produce a posterior distribution for the unknown that incorporates the answer you received. Typically, we consider an answer informative if it leads to a large change in our beliefs (change from prior to posterior), or, a large reduction in our uncertainty regarding the unknown.

Often, we are allowed to ask questions sequentially. At each stage we can adopt our posterior following the n^{th} question, answer pair (q_n, a_n) as our prior when asking the $n+1^{st}$ question. Since your posterior will depend on both the questions previously asked, and the answers previously given, a recursive procedure for selecting questions will specify a decision tree that guides the sequence of questions asked based on the sequence of answers received.

Viewing the search procedure as the specification of a decision tree reveals a duality with coding. Suppose that you can only ask Yes/No questions. If you fix your decision tree over the questions you ask, and observe a string of answers that point to a specific unknown (e.g. "Y, N, Y, Y, N, N, N , Y" imply $\Theta = \theta$) you could assign that unknown the string "YNYYNNNY" that specifies the path through the tree pointing to it. Then, the tree specifies an encoding function c that, when given a value of the unknown, returns the string of answers that would specify it, e.g. $c(\theta) = YNYYNNNY$. If you adopt any set of d-ary questions (questions with d possible answers), then the string may be converted into a d-ary sequence. For example, we could use 1 for Y and 0 for N, then assign $c(\theta) = 10110001$. This might correspond to the sequence: "Is an animal. Is not a mammal. Is a bird. Is flightless. Does not live in Antarctica. Does not live in Africa. Does not share a name with a fruit. Is an emu."

Complete information codes that aim for optimal compression attempt to find an encoding that minimizes the expected length of the code $c(\Theta)$ when $\Theta \sim p_\Theta$. Since the length of the code is the number of questions asked, this is the same as searching for a question procedure that is expected to terminate as quickly as possible.

In this project you will experiment with two essential heuristics for designing optimal, or near-to-optimal codes/search procedures. These are outlined below:

- **Fano Code:** "*At each stage, select the question whose answer is expected to yield the most information.*"

  This is a greedy search procedure that builds the code forward from the root by iteratively selecting the maximally informative question. It aims to maximize the mutual information between the answer to each question, and the unknown. It is mildly suboptimal.

- **Huffman Code:** "*At any given stage, never ask the question that is expected to yield the least information.*"

  Given a finite set of possible unknowns, this amounts to the heuristic, "*always ask the lest informative question last.*" Once a question has been posed, the sets of unknowns distinguished by the question can be grouped, and the heuristic can be iterated. This procedure builds the tree recursively from the leaves, iteratively grouping the least likely remaining sets of outcomes.

The Huffman code is information-theoretic optimal for finite sets of unknowns and questions with deterministic answers conditional on the unknown. When the answer is nondeterministic, no set of outcomes can be exactly grouped or split by a particular question,

and most finite question sequences cannot terminate in exact recovery of the unknown with complete certainty. In this case, it is impossible to build the tree from its leaves without coarse graining the space of unknowns (converting from an infinite set of unknowns to a finite set by rounding), or by defining a sufficient certainty at which the process stops. As a result, Fano's heuristic is more popular given an infinite set of unknowns, or imprecise measurements. Measurement procedures designed to maximize the expected information gained by an observation are popular in scientific settings where there is a large space of possible experiments to run, and a well defined statistical model relating experimental outcomes to the unknown.

For a deeper introduction to these problems, and examples, see Chapter 5 from Thomas and Cover's *Elements of Information Theory* (Second Edition).

Example Problems

For this project, you will practice implementing both Fano and Huffman codes for three toy search problems. The first two are associated with famous word games. The third is the Kalman filtering problem introduced in Lab 4. The two games are outlined below:

- **Twenty-Questions:** Your goal is to guess an unknown word drawn from a fixed dictionary. You ask questions in stages about the unknown word. All questions must be Yes/No questions, and will return answers Yes or No deterministically given the unknown word. You aim to resolve the unknown as quickly as possible (in fewer than 20 questions). For this project, we will assume you have a prior distribution over the possible words, and aim to select a decision tree over questions that minimizes the expected number of questions needed to discover the secret word.

- **Wordle:** Wordle is a word guessing-game owned and hosted by the New York Times. In Wordle, the player aims to guess an unknown, five-letter word. Their set of questions consist of other five-letter words. When they propose a five-letter word, they are told whether a given letter appears in the true word, and, if it appears in the true word in the place it appeared in the proposed word. For a description of the rules, see this NYT article. To test a Wordle strategy, go to Wordledictionary.com.

  Our goal is to select a decision tree over five-letter words that resolves the unknown word as quickly as possible. For a primer on optimal strategies in Wordle see this webpage, or 3blue1brown's video on Wordle.

Resources:

In the Zip folder uploaded on Ed, you will find:

- **Unigram_freq.csv:** A comma separated array containing two columns: (1) a list of 330,000 words sorted in decreasing order of frequency (in internet use), (2) a list of counts of each word (in the associated trawl of the internet).

- **Worlde_word_list.txt:** A text document containing the standard Wordle dictionary. This dictionary is available here. A larger list that differentiates the list of possible search words and possible true words is available here.

You may also find it helpful to look up packages containing standard word-embedding packages (for instance, see this primer on standard word embeddings).

Prompts

Complete **three of the following four** tasks. You *must* complete the first and fourth tasks (*Twenty Questions: Unconstrained Questions* and *Kalman: Noisy Observations*.) You may choose whether to complete the second or third prompt.

Twenty-Questions: Unconstrained Questions

First, let's practice designing an optimal search/coding procedure for the twenty-questions game with no constraints on the set of possible questions. If we place no constraints on the set of possible questions, than any $q \in \mathcal{Q}$ can be specified by selecting a set of words, $\mathcal{Y}$, and its complement $\mathcal{N}$, such that $A|\theta, q = 1$ if $\theta \in \mathcal{Y}$, and 0 otherwise.

(a) (5 points) Trim the Unigram_freq.csv array, isolating the most common 3,000 words in the list. Then construct:

- A Huffman code by iteratively grouping the two least common words in the list. Proceed by first grouping the two least common items. Then group these into a set, and assign the set the frequency equal to the sum of the frequencies of the two grouped words. Assign the set a key, remove the two words from the list, add the key to the list, then resort the list in descending order. Repeat this process iteratively to build the decision tree. See Thomas and Cover Chapter 5.6 for reference.
- A Fano code by sorting the words in descending order, then splitting the list into two halves. This can be done by normalizing the word frequencies, then computing the cumulative distribution function for the words when sorted in order of decreasing frequency. Find the division point such that the CDF is as close to 0.5 as possible. Once divided into two sets, repeat the procedure within each set. See Thomas and Cover Chapter 5.7 for reference.

(b) (1 point) How many Yes/No questions would it take, on average, to recover the unknown word if it is drawn from the top 3,000 words in the Unigram_freq.csv list with prior probabilities proportional to the observed word frequencies? How much less efficient is Fano's approach than Huffman's?

(c) (1 point) Compute the Shannon entropy (base 2) in the prior distribution over the top 3,000 words. This is a lower bound on the expected number of questions it would take any procedure to resolve an unknown word. This bound is achievable per encoded word for coding procedures that attempt to encode multiple independent samples from the prior at once. How far are your codes from optimal?

Twenty-Questions: Constrained Questions

The methods you developed above are practical for coding/look-up problems that are implemented in a computer, and that do not place realistic constraints on the ensemble of questions that can be asked.

Let's try implementing a heuristic procedures that attempts to imitate Fano's approach (greedily select the maximally informative question) while placing some constraints on the questions that we can ask. In the following exercises, you will use a word-embedding to map our dictionary into an embedding space. Then, we will consider all questions that can be

formed by performing a linear measurement (computing a dot-product) in the embedding space. If the embeddings assign meaningful coordinates to the embedded words, then these questions better approximate the sorts of questions available to humans.

(a) (3 points) Find and implement a word embedding that embeds based on cosine-similarity (see Word2Vec and GloVe for examples). You do not need to train your own. You can find trained versions available on github, python, or in some libraries (for instance: Matlab's implementation and embedding.

(b) (5 points) Write a code that:

1. Projects the top 3,000 words onto the unit sphere in the embedding space by normalizing their embeddings. Discard any words that cannot be embedded. Weight each word by its prior probability in the reduced list, using the frequencies in "Unigram-freq.csv" as the prior before eliminating words.
2. Iteratively identifies a hyperplane passing through the origin that divides the point cloud of embedded words into two sets, each containing roughly 50% of the current probability mass over possible words.[1]
3. Asks the question, "Is the unknown word on side A or B of the hyperplane?" by computing a dot product between the embedded word and the normal vector to the hyperplane, then checking its sign.
4. Drops all words from the list appearing on the wrong side of the hyperplane.
5. Iterates the search procedure until the unknown word is resolved.

(c) (3 points) Estimate, by sampling, the expected number of questions needed to resolve the unknown word. How much less efficient is this procedure? Propose an alternate scheme for selecting the hyerplane orientation (i.e. the linear measurement) which might lead to faster method, or, explain why you think your current scheme is near-to-optimal given the constraint on the space of questions allowed.

Wordle: Structured Observations

Twenty-Questions offers an easy observation model. Wordle offers a more interesting example by constraining the set of available questions to five-letter words, and providing a more structured likelihood model. Let's try to design a Wordle strategy based on iteratively selecting the most informative question.

(a) (1 point) First, create two prior distributions over the words in the Wordle dictionary. In the first, assume that the words are uniformly selected. In the second, assume that they are selected according to the frequencies provided in the Unigram_freq.csv. Look up the frequency of each word in the Wordle dictionary and drop, from both prior distributions, any word that does not appear in Unigram_freq.csv.

(b) (6 points) Write a code that can, given a prior distribution over the possible unknown words:

[1] A heuristic approch: compute the covariance in embedded coordinates, then select the hyperplane perpendicular to the direction of largest variance.

1. Search through all pairs of possible search words (questions) and unknown words we are trying to guess. Make sure your search is restricted to unknown words that are possible given the current prior. For each pair, compute the answer returned if the search word is proposed, and the unknown word is the true hidden word.
2. For each search word, identify all possible answers, the prior probability of each answer, and the collection of possible true, but unknown words that could have generated that answer.
3. For each (search word, answer) pair, compute the posterior distribution over the unknown true words given the prior and the returned answer.
4. Compute the Shannon entropy in the posterior given each possible search word, answer pair.
5. Compute the expected Shannon entropy in the posterior for each search term by averaging the posterior entropy after each answer over the possible answers.
6. Select the search term with the smallest expected posterior entropy.

(c) (1 point) Implement your code to derive an optimal search procedure for both priors.

(d) (2 points) Compute the expected number of search words needed to resolve a draw from both priors. Which prior makes for an easier game? Try to explain your observation.

(e) (1 point) Compare the first couple words in the decision tree proposed by both methods. Are they similar? If not, how do they differ? Does the difference sensibly reflect the difference in the priors?

Kalman: Noisy Answers

In many real-world experimental design problems our unknowns are continuously distributed, and our measurements are noisy. The best procedures in these settings typically involve selecting from a restricted set of possible questions in order to maximize the expected information gain per question asked. For this part, return to the Kalman filtering method introduced in Lab 4. This method assumes that, at each stage, the unknown is normally distributed, and, we can observe a noisy linear measurement whose noise is multivariate normal.

Assume that x is an unknown state vector drawn from a multivariate normal distribution and m is a measurement vector with as many entries as x. Assume that all measurements take the form $y = m \cdot x + \zeta$ where $\zeta \sim \mathcal{N}(0, s(m) \times \sigma_y^2)$ where $s(m)$ determines how the noise scales with the size of the measurement vector. The set of available questions $\mathcal{Q}$ consists of all measurements available when m is drawn from a set of available measurement vectors $\mathcal{M}$.

(a) (2 points) First, assume that $\mathcal{M}$ is the unit sphere and that $s(m) = \|m\|_2{}^2$ where $\|\cdot\|_2$ is the Euclidean norm. Write a code that:

1. Can compute the unit vector m which maximizes the expected information gained by each measurement (see HW 2).
2. Iteratively adopts the measurement that maximizes the expected information gained inside the Kalman filters used in Lab 4.

(b) (2 points) How much more effective is this scheme than the fixed measurement schemes used in Lab 4 originally? Try re-running your experiments from Lab 4 to compare the

time it takes the filter to converge (in the Harmonic oscillator case), and its accuracy (in the Lorenz example).

(c) (1 point) Compute the "entropy rate" of the process. This equals, to a reasonable approximation, the differential entropy in the process noise distribution plus the log determinant of the Jacobian used in each update step, averaged across the collection of steps. Compute the "information rate" of the measurement process, that is, the average information gain per measurement, averaged over all measurements taken. See HW 2 for the relevant formula. For both models, check whether the information rate is greater than or less than the entropy rate. Does your answer explain whether the filter converges?

(d) (3 points) Usually we are not free to make arbitrary linear measurements. Instead, we are restricted to a set of measurements drawn from a discrete set. Let $\mathcal{M}$ consist only of the the canonical basis vectors $e_1 = [1, 0, \ldots]$, $e_2 = [0, 1, \ldots]$, and their unit linear combinations $e_1 + e_2$, $e_1 + e_2 + e_3$, $e_1 - e_2$, etc. Assume that $s(m) = \|m\|_1$.

Write a code that can:

1. Compute the vector $m \in \mathcal{M}$ that maximizes the expected information gained by each measurement (see HW 2). The number of possible measurement vectors is small in all cases, so you can use brute force here.
2. Iteratively adopts the measurement that maximizes the expected information gained inside the Kalman filters used in Lab 4.
3. Reproduces the visuals used in Lab 4, but colors the scatter points representing observations according to the measurement vector used. Add a legend mapping scatter color to the choice of measurement vector.

(e) (2 points) Apply this scheme to the Kalman filter examples in Lab 4, and compare your results to the results you saw when allowing any vector drawn from the unit sphere. Try to explain your observations. It may help to repeat the entropy rate vs information rate comparison you made before, using the constrained set of measurement vectors.

(f) (1 point) How much less effective is this method if you do not allow unit linear combinations (you only allow the basis vectors $\pm e_j$)? How does the information rate vary with the space of available measurements?

(g) (1 point) Comment on any patterns you notice regarding the choice of measurement vector.